Docket No. AUS920030543US1

METHOD AND APPARATUS FOR PROVIDING PRE AND POST HANDLERS FOR RECORDING EVENTS

CROSS REFERENCE TO RELATED APPLICATIONS

The present invention is related to the following applications entitled "Method and Apparatus for Counting Instruction Execution and Data Accesses", serial no. _____, attorney docket no. AUS920030477US1, filed on September 30, 2003; "Method and Apparatus for Selectively Counting Instructions and Data Accesses", serial no. _____, attorney docket no. AUS920030478US1, filed on September 30, 2003; "Method and Apparatus for Generating Interrupts Upon Execution of Marked Instructions and Upon Access to Marked Memory Locations", serial no. _____, attorney docket no. AUS920030479US1, filed on September 30, 2003; "Method and Apparatus for Counting Data Accesses and Instruction Executions that Exceed a Threshold", serial no. _____, attorney docket no. AUS920030480US1, filed on September 30, 2003; "Method and Apparatus for Counting Execution of Specific Instructions and Accesses to Specific Data Locations", serial no. _____, attorney docket no. AUS920030481US1, filed on September 30, 2003; "Method and Apparatus for Debug Support for Individual Instructions and Memory Locations", serial no. _____, attorney docket no. AUS920030482US1, filed on September 30, 2003; "Method and Apparatus to Autonomically Select Instructions for Selective Counting", serial no. _____, attorney docket no. AUS920030483US1, filed on September 30, 2003;

"Method and Apparatus to Autonomically Count Instruction Execution for Applications", serial no. _____, attorney docket no. AUS920030484US1, filed on September 30, 2003; "Method and Apparatus to Autonomically Take an Exception on Specified Instructions", serial no. _____, attorney docket no. AUS920030485US1, filed on September 30, 2003; "Method and Apparatus to Autonomically Profile Applications", serial no. _____, attorney docket no. AUS920030486US1, filed on September 30, 2003; "Method and Apparatus for Counting Instruction and Memory Location Ranges", serial no. _____, attorney docket no. AUS920030487US1, filed on September 30, 2003; "Method and Apparatus for Qualifying Collection of Performance Monitoring Events by Types of Interrupt When Interrupt Occurs", serial no. _____, attorney docket no. AUS920030540US1, filed on _____; and "Method and Apparatus for Counting Interrupts by Type", serial no. _____, attorney docket no. AUS920030541US1, filed on _____. All of the above related applications are assigned to the same assignee, and incorporated herein by reference.

BACKGROUND OF THE INVENTION

1. Technical Field:

The present invention relates generally to an improved data processing system and, in particular, to a method and system for monitoring performance of the processor in a data processing system when an interrupt occurs. Still more particularly, the present invention relates to a method, apparatus, and computer instructions for providing pre handlers and post handlers to record events.

2. Description of Related Art:

A typical data processing system utilizes processors to execute a set of instructions in order to perform a certain task, such as reading a specific character from the main memory. However, as the number of tasks required to be executed by the processor increases, the efficiency of the processor's access patterns to memory and the characteristics of such access become important factors for engineers who want to optimize the system.

Currently, the prior art contains mechanisms that can count occurrences of software-selectable events, such as, for example, cache misses, instructions executed, I/O data transfer request, and the time a given process may take to execute within a data processing system. One such mechanism is a performance monitor. A performance monitor monitors selected characteristics for system

analysis by determining a machine's state at a particular time. This analysis provides information of how the processor is used when instructions are executed and its interaction with the main memory when data is stored. This analysis may also be used to determine if application code changes, such as a relocation of branch instructions and memory access, to further optimize the performance of a system are necessary. In addition, the performance monitor may provide the amount of time that has passed between events in a processing system. The performance monitor counts events that may be used by engineers to analyze system performance. Moreover, data regarding how the processor accesses the data processing system's level 1 and level 2 cache, and main memory may be gathered by the performance monitor in order to identify performance bottlenecks that are specific to a hardware or software environment.

In addition to the performance monitor described above, an interrupt processing unit may be used to record events such as, for example, instruction execution, branch events, or system events when an interrupt occurs. An interrupt occurs when a device, such as a mouse or keyboard, raises an interrupt signal to notify the processor that an event has occurred. When the processor accepts an interrupt request, the processor completes its current instruction and passes the control to an interrupt handler. The interrupt handler executes an interrupt service routine that is associated with the interrupt. An interrupt may also be caused by a specific machine language operation code, for example Motorola

68000's TRAP, a product from Motorola, Inc. In this case, an unexpected software condition such as divide by zero causes the processor to store the current state, store identifying information about the particular interrupt and pass control to an interrupt handler that handles this unexpected software condition.

However, the performance monitor above must modify the application program at run time in order to record precise performance trace data, such as the number of instructions executed during interrupt processing. Therefore, it would be advantageous to have an improved method, apparatus, and computer instructions for providing pre and post handlers to record precise performance data for events occurring before entering and immediately after exiting an interrupt handler without modifying underlying application program.

SUMMARY OF THE INVENTION

The present invention provides a method, apparatus, and computer instructions for providing pre and post handlers to record events when an interrupt occurs. The pre and post handlers allow logging of trace records along with timestamps associated with performance monitoring events to be recorded in order to provide the user with more fine-grained performance data.

In a preferred embodiment, the mechanism of the present invention provides pre and post handlers to record the occurrence of performance monitoring events when a branch instruction is executed. The pre and post handlers are used with a "trap on branch", i.e. a trap, or interrupt, being processed when a branch instruction is executed, to produce an instruction trace which includes the 'from' address of where the branch is taken and may include the 'to' address of where the branch branches to. It should be clear that information could be compressed in various ways to minimize the amount of information to be recorded. The pre and post handlers record performance monitoring events occurring prior to and immediately after taking the branch.

In an alternative embodiment, before the processor fetches instructions from the interrupt handler when an interrupt occurs, the mechanism of the present invention allows the pre handler to log trace records prior to entering the interrupt handler. The events recorded provide the state of the system when entering an interrupt handler.

When the interrupt handler completes the interrupt service routine, the mechanism of the present inventions allows the post handler to record events and low level information, such as the number of instructions executed for an interrupt, before returning to normal execution. This low-level information may provide the state of the system when exiting an interrupt.

These and other features and advantages will be discussed in, or will become apparent to those of ordinary skill in the art in view of, the following detailed description of the preferred embodiments.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as a preferred mode of use, further objectives and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

Figure 1 is an exemplary block diagram of a data processing system in which the present invention may be implemented;

Figure 2 is an exemplary block diagram of a processor system for processing information in accordance with a preferred embodiment of the present invention;

Figure 3 is an exemplary diagram illustrating components for recording events of an interrupt using pre and post handler in accordance with a preferred embodiment of the present invention;

Figure 4 is an exemplary diagram illustrating an example interrupt description table (IDT) in accordance with a preferred embodiment of the present invention;

Figure 5 is a flowchart outlining an exemplary process for recording events of an interrupt using pre and post handlers in accordance with a preferred embodiment of the present invention; and

Figure 6 is a flowchart outlining an exemplary process for logging trace records when a branch instruction is executed with trap on branch using pre and post handlers in accordance with a preferred embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The mechanism of the present invention provides pre and post handlers to log trace records, including the 'from' address of where an interrupt occurs or where a branch instruction is executed. The pre and post handlers record events, such as performance monitoring events, prior to entering and after exiting an interrupt handler or taking a branch. By logging these trace records, engineers may identify values of performance monitoring events at a particular instant in time. These records may assist engineers in isolating events that occur during normal execution of the system from events that occur in the system when an interrupt is handled or when a branch is taken.

The present invention may be implemented in a computer system. The computer system may be a stand-alone computing device, a client or server computing device in a client-server environment that is interconnected over a network, or the like. **Figure 1** provides an exemplary diagram of a computing device in which aspects of the present invention may be implemented. **Figure 1** is only exemplary and no limitation on the structure or organization of the computing devices on which the present invention may be implemented is asserted or implied by the depicted example.

With reference now to **Figure 1**, an exemplary block diagram of a data processing system is shown in which the present invention may be implemented. Client **100** is an example of a computer, in which code or instructions

implementing the processes of the present invention may be located. Client **100** employs a peripheral component interconnect (PCI) local bus architecture. Although the depicted example employs a PCI bus, other bus architectures such as Accelerated Graphics Port (AGP) and Industry Standard Architecture (ISA) may be used. Processor **102** and main memory **104** are connected to PCI local bus **106** through PCI bridge **108**. PCI bridge **108** also may include an integrated memory controller and cache memory for processor **102**. Additional connections to PCI local bus **106** may be made through direct component interconnection or through add-in boards.

In the depicted example, local area network (LAN) adapter **110**, small computer system interface SCSI host bus adapter **112**, and expansion bus interface **114** are connected to PCI local bus **106** by direct component connection. In contrast, audio adapter **116**, graphics adapter **118**, and audio/video adapter **119** are connected to PCI local bus **106** by add-in boards inserted into expansion slots. Expansion bus interface **114** provides a connection for a keyboard and mouse adapter **120**, modem **122**, and additional memory **124**. SCSI host bus adapter **112** provides a connection for hard disk drive **126**, tape drive **128**, and CD-ROM drive **130**. Typical PCI local bus implementations will support three or four PCI expansion slots or add-in connectors.

An operating system runs on processor **102** and is used to coordinate and provide control of various components within data processing system **100** in **Figure 1**. The operating system may be a commercially available operating system such as Windows XP, which is available from

Microsoft Corporation. An object oriented programming system such as Java may run in conjunction with the operating system and provides calls to the operating system from Java programs or applications executing on client **100**. "Java" is a trademark of Sun Microsystems, Inc. Instructions for the operating system, the object-oriented programming system, and applications or programs are located on storage devices, such as hard disk drive **126**, and may be loaded into main memory **104** for execution by processor **102**.

Those of ordinary skill in the art will appreciate that the hardware in **Figure 1** may vary depending on the implementation. Other internal hardware or peripheral devices, such as flash read-only memory (ROM), equivalent nonvolatile memory, or optical disk drives and the like, may be used in addition to or in place of the hardware depicted in **Figure 1**. Also, the processes of the present invention may be applied to a multiprocessor data processing system.

For example, client **100**, if optionally configured as a network computer, may not include SCSI host bus adapter **112**, hard disk drive **126**, tape drive **128**, and CD-ROM **130**. In that case, the computer, to be properly called a client computer, includes some type of network communication interface, such as LAN adapter **110**, modem **122**, or the like. As another example, client **100** may be a stand-alone system configured to be bootable without relying on some type of network communication interface, whether or not client **100** comprises some type of network communication interface. As a further example, client

100 may be a personal digital assistant (PDA), which is configured with ROM and/or flash ROM to provide non-volatile memory for storing operating system files and/or user-generated data. The depicted example in **Figure 1** and above-described examples are not meant to imply architectural limitations.

The processes of the present invention are performed by processor **102** using computer implemented instructions, which may be located in a memory such as, for example, main memory **104**, memory **124**, or in one or more peripheral devices **126-130**.

Turning next to **Figure 2**, an exemplary block diagram of a processor system for processing information is depicted in accordance with a preferred embodiment of the present invention. Processor **210** may be implemented as processor **102** in **Figure 1**.

In a preferred embodiment, processor **210** is a single integrated circuit superscalar microprocessor. Accordingly, as discussed further herein below, processor **210** includes various units, registers, buffers, memories, and other sections, all of which are formed by integrated circuitry. Also, in the preferred embodiment, processor **210** operates according to reduced instruction set computer ("RISC") techniques. As shown in **Figure 2**, system bus **211** is connected to a bus interface unit ("BIU") **212** of processor **210**. BIU **212** controls the transfer of information between processor **210** and system bus **211**.

BIU **212** is connected to an instruction cache **214** and to data cache **216** of processor **210**. Instruction cache **214** outputs instructions to sequencer unit **218**. In response to such instructions from instruction cache **214**, sequencer unit **218** selectively outputs instructions to other execution circuitry of processor **210**.

In addition to sequencer unit **218**, in the preferred embodiment, the execution circuitry of processor **210** includes multiple execution units, namely a branch unit **220**, a fixed-point unit A ("FXUA") **222**, a fixed-point unit B ("FXUB") **224**, a complex fixed-point unit ("CFXU") **226**, a load/store unit ("LSU") **228**, and a floating-point unit ("FPU") **230**. FXUA **222**, FXUB **224**, CFXU **226**, and LSU **228** input their source operand information from general-purpose architectural registers ("GPRs") **232** and fixed-point rename buffers **234**. Moreover, FXUA **222** and FXUB **224** input a "carry bit" from a carry bit ("CA") register **239**. FXUA **222**, FXUB **224**, CFXU **226**, and LSU **228** output results (destination operand information) of their operations for storage at selected entries in fixed-point rename buffers **234**. Also, CFXU **226** inputs and outputs source operand information and destination operand information to and from special-purpose register processing unit ("SPR unit") **237**.

FPU **230** inputs its source operand information from floating-point architectural registers ("FPRs") **236** and floating-point rename buffers **238**. FPU **230** outputs

results (destination operand information) of its operation for storage at selected entries in floating-point rename buffers **238**.

In response to a Load instruction, LSU **228** inputs information from data cache **216** and copies such information to selected ones of rename buffers **234** and **238**. If such information is not stored in data cache **216**, then data cache **216** inputs (through BIU **212** and system bus **211**) such information from a system memory **239** connected to system bus **211**. Moreover, data cache **216** is able to output (through BIU **212** and system bus **211**) information from data cache **216** to system memory **239** connected to system bus **211**. In response to a Store instruction, LSU **228** inputs information from a selected one of GPRs **232** and FPRs **236** and copies such information to data cache **216**.

Sequencer unit **218** inputs and outputs information to and from GPRs **232** and FPRs **236**. From sequencer unit **218**, branch unit **220** inputs instructions and signals indicating a present state of processor **210**. In response to such instructions and signals, branch unit **220** outputs (to sequencer unit **218**) signals indicating suitable memory addresses storing a sequence of instructions for execution by processor **210**. In response to such signals from branch unit **220**, sequencer unit **218** inputs the indicated sequence of instructions from instruction cache **214**. If one or more of the sequence of instructions is not stored in instruction cache **214**, then instruction

cache **214** inputs (through BIU **212** and system bus **211**) such instructions from system memory **239** connected to system bus **211**.

In response to the instructions input from instruction cache **214**, sequencer unit **218** selectively dispatches the instructions to selected ones of execution units **220**, **222**, **224**, **226**, **228**, and **230**. Each execution unit executes one or more instructions of a particular class of instructions. For example, FXUA **222** and FXUB **224** execute a first class of fixed-point mathematical operations on source operands, such as addition, subtraction, ANDing, ORing and XORing. CFXU **226** executes a second class of fixed-point operations on source operands, such as fixed-point multiplication and division. FPU **230** executes floating-point operations on source operands, such as floating-point multiplication and division.

As information is stored at a selected one of rename buffers **234**, such information is associated with a storage location (e.g. one of GPRs **232** or carry bit(CA) register **242**) as specified by the instruction for which the selected rename buffer is allocated. Information stored at a selected one of rename buffers **234** is copied to its associated one of GPRs **232** (or CA register **242**) in response to signals from sequencer unit **218**. Sequencer unit **218** directs such copying of information stored at a selected one of rename buffers **234** in response to "completing" the instruction that generated the information. Such copying is called "writeback."

As information is stored at a selected one of rename buffers **238**, such information is associated with one of FPRs **236**. Information stored at a selected one of rename buffers **238** is copied to its associated one of FPRs **236** in response to signals from sequencer unit **218**. Sequencer unit **218** directs such copying of information stored at a selected one of rename buffers **238** in response to "completing" the instruction that generated the information.

Processor **210** achieves high performance by processing multiple instructions simultaneously at various ones of execution units **220**, **222**, **224**, **226**, **228**, and **230**. Accordingly, each instruction is processed as a sequence of stages, each being executable in parallel with stages of other instructions. Such a technique is called "pipelining." In a significant aspect of the illustrative embodiment, an instruction is normally processed as six stages, namely fetch, decode, dispatch, execute, completion, and writeback.

In the fetch stage, sequencer unit **218** selectively inputs (from instruction cache **214**) one or more instructions from one or more memory addresses storing the sequence of instructions discussed further hereinabove in connection with branch unit **220**, and sequencer unit **218**.

In the decode stage, sequencer unit **218** decodes up to four fetched instructions.

In the dispatch stage, sequencer unit **218** selectively dispatches up to four decoded instructions to selected (in response to the decoding in the decode

stage) ones of execution units **220**, **222**, **224**, **226**, **228**, and **230** after reserving rename buffer entries for the dispatched instructions' results (destination operand information). In the dispatch stage, operand information is supplied to the selected execution units for dispatched instructions. Processor **210** dispatches instructions in order of their programmed sequence.

In the execute stage, execution units execute their dispatched instructions and output results (destination operand information) of their operations for storage at selected entries in rename buffers **234** and rename buffers **238** as discussed further hereinabove. In this manner, processor **210** is able to execute instructions out-of-order relative to their programmed sequence.

In the completion stage, sequencer unit **218** indicates an instruction is "complete." Processor **210** "completes" instructions in order of their programmed sequence.

In the writeback stage, sequencer **218** directs the copying of information from rename buffers **234** and **238** to GPRs **232** and FPRs **236**, respectively. Sequencer unit **218** directs such copying of information stored at a selected rename buffer. Likewise, in the writeback stage of a particular instruction, processor **210** updates its architectural states in response to the particular instruction. Processor **210** processes the respective "writeback" stages of instructions in order of their programmed sequence. Processor **210** advantageously merges an instruction's completion stage and writeback stage in specified situations.

In the illustrative embodiment, each instruction requires one machine cycle to complete each of the stages of instruction processing. Nevertheless, some instructions (e.g., complex fixed-point instructions executed by CFXU **226**) may require more than one cycle. Accordingly, a variable delay may occur between a particular instruction's execution and completion stages in response to the variation in time required for completion of preceding instructions.

Completion buffer **248** is provided within sequencer **218** to track the completion of the multiple instructions which are being executed within the execution units. Upon an indication that an instruction or a group of instructions have been completed successfully, in an application specified sequential order, completion buffer **248** may be utilized to initiate the transfer of the results of those completed instructions to the associated general-purpose registers.

In addition, processor **210** also includes performance monitor unit **240**, which is connected to instruction cache **214** as well as other units in processor **210**. Operation of processor **210** can be monitored utilizing performance monitor unit **240**, which in this illustrative embodiment is a software-accessible mechanism capable of providing detailed information descriptive of the utilization of instruction execution resources and storage control. Although not illustrated in **Figure 2**, performance monitor unit **240** is coupled to each functional unit of processor **210** to permit the monitoring of all aspects of the operation of processor **210**, including, for example,

reconstructing the relationship between events, identifying false triggering, identifying performance bottlenecks, monitoring pipeline stalls, monitoring idle processor cycles, determining dispatch efficiency, determining branch efficiency, determining the performance penalty of misaligned data accesses, identifying the frequency of execution of serialization instructions, identifying inhibited interrupts, and determining performance efficiency. The events of interest also may include, for example, time for instruction decode, execution of instructions, branch events, cache misses, and cache hits.

Performance monitor unit **240** includes an implementation-dependent number (e.g., 2-8) of counters **241-242**, labeled PMC1 and PMC2, which are utilized to count occurrences of selected events. Performance monitor unit **240** further includes at least one monitor mode control register (MMCR). In this example, two control registers, MMCRs **243** and **244** are present that specify the function of counters **241-242**. Counters **241-242** and MMCRs **243-244** are preferably implemented as SPRs that are accessible for read or write via MFSPR (move from SPR) and MTSPR (move to SPR) instructions executable by CFXU **226**. However, in one alternative embodiment, counters **241-242** and MMCRs **243-244** may be implemented simply as addresses in I/O space. In another alternative embodiment, the control registers and counters may be accessed indirectly via an index register. This embodiment is implemented in the IA-64 architecture in processors from Intel Corporation.

Additionally, processor **210** also includes interrupt unit **250**, which is connected to instruction cache **214**. Additionally, although not shown in **Figure 2**, interrupt unit **250** is connected to other functional units within processor **210**. Interrupt unit **250** may receive signals from other functional units and initiate an action, such as starting an error handling or trap process. In these examples, interrupt unit **250** is employed to generate interrupts and exceptions that may occur during execution of a program.

The present invention provides a method, apparatus, and computer instructions for providing pre and post handlers to record events. The pre and post handlers may perform different operations to obtain different computer program execution metric information for use by a performance analysis tool in providing analysis of the computer program's performance during execution. The pre handler is intended to be used to obtain information about computer program execution metrics as they are prior to execution of interrupt handling routines. The post handler is intended to be used to obtain information about what occurs during the handling of an interrupt, i.e. during execution of the interrupt handling routines.

A trap, i.e. a piece of code that executes when a particular condition has occurred, such as a when a particular interrupt or exception has been generated, etc also is referred to herein as a trap or an interrupt service routines.

With the present invention, as part of handling this trap, the processor of the present invention calls a pre

handler prior to calling the trap or interrupt handling routines. That is, when the trap instruction generates an interrupt, or otherwise attempts to transfer control to a trap handling routine, the processor redirects this call to a pre handler which may accumulate or otherwise obtain performance monitoring metric information from performance monitor mechanisms associated with the computer program. This information may then be stored for identifying the state of the execution of the computer program prior to the trap or interrupt handler having been executed.

Similarly, when a trap occurs and the trap or interrupt handling routine has been executed, the post handler may be executed after execution of the trap or interrupt handling routine to obtain information about what occurred during execution of the trap or interrupt handling routine. For example, the post handler may add the number of instructions or cache misses that occurred in the interrupt handler itself to accumulate a total value that can be output to the performance analysis tool for use during performance analysis of the computer program. In order to perform different operations, the pre and post handlers process performance monitor metric information, hereafter referred to as "performance information", using values of events that occurred and timestamps associated with the events.

In addition, the pre and post handlers may log trace records. A trace record may include a 'from' address of the instruction where an event occurs as well as counts for a selected event, and a time stamp identifying when

the trace record is being written. Events recorded may include performance monitoring events occurring before entering and after exiting an interrupt service routine. The pre and post handlers are instruction routines that log trace records or process information about the state of the machine. Trace records may also include information read from performance monitoring counters, such as counters **241-242** in **Figure 2**, and timestamps associated for each event. Examples of performance monitoring events are the number of instructions executed, the number of cache misses, the number of table lookaside buffer (TLB) misses, etc.

The pre and post handlers accumulate trace records for events that occur prior to and during the execution of the interrupt service routine, or trap/interrupt handling routine, by recording the values of performance monitor counters before executing the handler and recording the values of the performance monitor counters right after executing the handler. The performance monitor may be programmed to stop counting events while the pre and post handlers are being executed.

The pre and post handlers of the present invention may be enabled or disabled. As a result, customized performance monitoring information may be obtainable by determining which pre and post handlers to enable or disable. In this way, a user may obtain performance information that is of particular interest to the problems of importance to that user. This permits a great deal of flexibility with regard to what performance

monitoring information is compiled and analyzed by pre and post handlers.

In addition, the pre and post handler routines may also be used with "trap on branch" instructions to produce a trace record. As touched upon above, a trap is a specialized piece of code in a program that occurs due to a particular condition in a running program.

A "trap on branch", otherwise referred to as a "branch trap", is a trap that occurs if a branch is taken. If the branch is taken, then the trap on branch handler receives control.

In a preferred embodiment of the present invention, one or more branches in the computer program being monitored are selected for monitoring. When one of these branch instructions selected for monitoring is executed, i.e. when the branch is taken, the processor sends a signal to the interrupt unit. Rather than fetching instructions from the normal trap handling routine, the interrupt unit notifies the processor to fetch instructions from the pre handler to log trace records prior to executing the trap handling routine and taking the branch, if the pre handler is enabled.

A trace record may include a 'from' memory address, which is the memory address of the branch instruction when the branch is taken. In addition, the trace record may also include a 'to' memory address, which is the memory address of where to branch to. Associated with these addresses may be performance monitoring information that may be obtained from performance monitoring devices and data structures, e.g., counters **241-242**. An example

of performance monitoring information that may be obtained and associated with the 'to' and/or 'from' and from addresses is selected counts of performance monitoring events, such as the number of cache misses that have occurred.

After the pre handler routine is executed, the normal trap service routine, or trap handler routine, is be executed. A post handler routine may also be associated with the trap that is placed in the program. This post handler routine is called after the normal trap service routine has finished executing and attempts to return control of the computer program to the normal code of the computer program. Upon completion of the normal trap service routine, and prior to returning to normal execution, the processor sends a signal to the interrupt unit indicating completion of the normal trap service routine. As a result, if the post handler is enabled, the interrupt unit notifies the processor to fetch instructions from the post handler in order to log trace records after taking the branch. Thus, the pre and post handlers allow the user to record low-level performance data prior to and immediately after execution of a trap/interrupt handling routine. One preferred application of this functionality is for the "trap on branch" conditions discussed above.

Normally, when an interrupt occurs, such as when a page fault occurs during program execution, the processor stops the current execution and begins fetching instructions from the address of the entry point to the interrupt handler. An interrupt handler includes

interrupt service routines that are fetched by the processor to handle an interrupt. The address of the entry point is stored in the interrupt descriptor table (IDT) which is a system table that associates each interrupt with a corresponding interrupt handler containing corresponding interrupt service routines.

However, in one exemplary embodiment of the present invention, in response to the occurrence of an interrupt event that causes an interrupt to be generated, the interrupt unit of the present invention receives an interrupt signal from the processor with associated metadata. This metadata may include handler flags, e.g., non-zero values that identify one of the following interrupt handlers: normal interrupt handler, pre handler, or post handler.

If a pre handler flag is set in the metadata, rather than fetching instructions from the normal interrupt handler as described above, the interrupt unit notifies the processor to fetch instructions from the pre handler routine that records trace data occurring prior to execution of the "normal" interrupt handler, at a particular instant of time. A "normal" interrupt handler, as the term is used in this description, refers to the interrupt handling routine that would execute based on the interrupt generated if the pre and/or post flags were not set.

The trace data generated by the pre handler routine may include a 'to' and/or 'from' address of the instruction where the interrupt occurs and selected

performance monitoring information. A timestamp may also be included in the trace record.

Once the pre handler routine is executed, the pre handler notifies the processor to fetch instructions from the normal interrupt handler. When the normal interrupt handler execution is complete, if the post handler flag is set, the interrupt unit of the present invention notifies the processor to fetch instructions from the post handler to record trace data for an interrupt at a particular instant of time after the normal interrupt handler.

In this way, the pre and post handlers allow trace data specific to a particular trap or interrupt at a particular instant of time to be recorded without making modification to any operating system routines.
Using the pre and post handlers, trace records may be logged prior to execution of a normal trap or interrupt handling routine, such as in response to the taking of a branch, and immediately after execution of a normal trap or interrupt handling routine. From the information generated by the pre and post handlers, a picture of what is happening in the computer program execution between the time when a branch instruction is executed to the time when a branch is taken, and between the time after a branch is taken and returning to normal execution, may be obtained.

In addition, performance trace data may be recorded at the entrance and exit of the normal trap or interrupt handling routine in order to identify changes that occur between the time when a particular interrupt occurs to

the start of the trap or interrupt handling routine. In addition, changes that occur between the end of the trap or interrupt handling routine and the interrupt return may also be identified. This information provides engineers with a tool to separate events that occur during normal execution of the system from events that occur when the system is interrupted or a branch is taken.

In yet another preferred embodiment, the pre and post handler routines may be implemented to perform other functions, such as handling overflow of the count used by the performance monitoring unit to count events. The performance monitoring unit may be implemented as performance monitoring unit **240** in **Figure 2**. The count may be stored either in the IDT or in a dedicated memory location outside of the IDT. Because an overflow may occur during counting of events, the pre and post handler routines may look up the value of the count periodically to check for overflow. In one embodiment, if the value of the count is about to wrap, the pre and post handler routines may signal the processor that an overflow will occur. In another embodiment, the pre and post handlers may include routines that handle the overflow, for example, by reading and resetting the count value.

Turning next to **Figure 3**, an exemplary diagram illustrating components for recording events of an interrupt using pre and post handler is depicted in accordance with a preferred embodiment of the present invention. In this example implementation, the central processing unit (CPU) **302** may be implemented as processor

210 in **Figure 2**. In a preferred embodiment, when a branch instruction is executed, a trap is executed by the application program. The trap notifies CPU **302** to generate and send a signal to interrupt unit **304** indicating that an exception has occurred, meaning the branch instruction is executed. Interrupt unit **304** notifies CPU **302** to fetch instructions from the pre handler to record trace data prior to taking a branch. The pre handler routine records trace data, which may include a 'from' address where the branch instruction is executed, the 'to' address of where to branch and performance information, e.g., count values, of selected performance events that occurred, prior to taking a branch.

After the execution of the trap handling routine, an indication of the completion of the trap handling routine is returned to the interrupt unit **304**. The interrupt unit **304** then notifies CPU **302** to fetch instructions from the post handler to record trace data prior to returning to normal execution.

In an alternative embodiment, when an interrupt occurs, such as when a cache miss occurs during program execution, CPU **302** sends a signal to interrupt unit **304** with associated metadata that identifies the type of interrupt, which is used to identify the particular interrupt handling routine that is to be executed to handle the interrupt. Interrupt unit **304** may then identify the address in the IDT and notifies the processor to fetch instructions from the interrupt handling routine at the address identified in the IDT

based on the interrupt type indicated in the metadata. This may involve fetching instructions from an address for a pre and/or post handler that is stored in association with the handler identified by the metadata, as described hereafter.

Turning next to **Figure 4**, an exemplary diagram illustrating an example interrupt description table (IDT) is depicted in accordance with a preferred embodiment of the present invention. In this example implementation, an interrupt descriptor table (IDT) **402** includes memory addresses **404**, interrupt types **406**, pre handler addresses **408** and post handler addresses **410**. Interrupt types **406** represented in IDT **402** are for illustrative purpose only.

In this example, when an interrupt occurs such as, for example, a Virtual Hash Page Table (VHPT) data fault interrupt, which is an interrupt associated Virtual Hash Page Table, the processor sends a signal identifying the occurrence of the interrupt to an interrupt unit, such as interrupt unit **304** in **Figure 3**. The interrupt unit checks metadata **400** associated with the signal to determine if pre or post handlers are enabled for handling the identified interrupt. In this example, the metadata is 01, which means the pre handler is enabled. If the metadata is 00, the normal interrupt handler is enabled; if the metadata is 10, the post handler is enabled. If the metadata is set to 11, both the pre handler and the post handler are enabled.

Whether the metadata is set to 01, 00, 10 or 11 may be determined based on, for example, a data structure associated with the performance monitoring applications

that are used to monitor the execution of the computer program. That is, a user may set whether pre handlers, post handlers, both, or neither are enabled for a particular type of event or for a particular trace, via user input to the performance monitoring application. This information may be stored in a data structure associated with the performance monitoring application which may be accessed to set registers associated with the processor indicating whether to enable pre handlers, post handlers, both or neither.

Alternatively, this information may be maintained in a data structure that is accessed by the processor or the interrupt unit each time there is a trap or interrupt generated to determine whether that particular trap or interrupt is associated with an enabled pre handler, post handler, both or neither. This data structure may be, for example, the IDT, a shadow IDT data structure, or other data structure. In another embodiment, a register may be used to indicate the address of a pre handler and a different register for a post handler. The value of zero means there is no pre or post handler and a non-zero value indicates the address of the pre or post handler. This approach allows one pre handle and one post handler for all traps. [Please note that is probably the preferred embodiment and should definitely be in the claims.]

If the pre handler is enabled, the interrupt unit identifies the pre handler routine address **408**, in this example 0x4000 **412**, from pre handler address field **408** of IDT **402** associated with the interrupt type indicated in

the received signal from the processor and executes the corresponding pre handler routine **422** starting at memory address 0x4000 **424** to record events.

Once pre handler routine **422** is executed, the interrupt unit identifies starting memory address **404** of the VHPT data fault interrupt service routine from memory address field **404** of IDT **402**, in this example 0x0000 **426** and notifies the processor to execute the normal interrupt service routine of the interrupt handler. Once the normal interrupt service routine is complete, the processor sends a signal to the interrupt unit **304** in **Figure 3**. The interrupt unit checks metadata **400**, which is associated with the signal sent by the processor in response to the interrupt occurring, to determine whether the post handler is also enabled.

If the post handler is enabled, the interrupt unit **304** identifies post handler routine starting address **410**, in this example 0x5000 **442**, from post handler address field **410** of IDT **402**. The interrupt unit then notifies the processor to execute corresponding post handler routine **440** starting at memory address 0x5000 **442** to record events. Once post handler routine **440** is executed, the interrupt unit then notifies the processor to return to the original instructions of the computer program that caused the interrupt.

Turning next to **Figure 5**, a flowchart outlining an exemplary process for recording events of an interrupt using pre and post handlers from the interrupt unit's perspective is depicted in accordance with a preferred embodiment of the present invention. The process begins

when the processor, in response to executing a trap instruction or an interrupt being generated, sends a signal with associated metadata to the interrupt unit (step **502**). Next, a determination is made as to whether the pre handler is enabled based on the metadata associated with the signal (step **504**) by examining the flag set in the metadata. If the pre handler is not enabled, the interrupt unit identifies the memory address of the normal interrupt handling routine, such as memory address **404** in **Figure 4**, corresponding to the interrupt type, such as interrupt type **406** in **Figure 4** (step **510**) in the IDT, such as IDT **402** in **Figure 4** and notifies the processor to execute the interrupt handling routine (step **512**).

If the pre handler is enabled, the interrupt unit identifies the starting address of the pre handler routine (step **506**) and notifies the processor to execute the pre handler routine (step **508**) to record events. The interrupt unit then notifies the processor the memory address of the interrupt handling routine corresponding to the interrupt type (step **510**) in the IDT, in order to execute the interrupt handling routine (step **512**).

Once the processor executes the interrupt handling routine, the interrupt unit then identifies the post handler address that corresponds to the interrupt type or IDT entry in the IDT (step **514**). A determination is made as to whether the post handler is enabled based on the metadata (step **516**). If the post handler is not enabled, the interrupt unit notifies the processor to return to

the original execution (step **522**). The process terminates thereafter.

If the post handler is enabled, the interrupt unit identifies the starting address of the post handler routine (step **518**). The interrupt unit then notifies the processor to execute the post handler routine (step **520**) to record events before returning to original execution (step **522**). The process then terminates thereafter.

Turning next to **Figure 6**, a flowchart outlining an exemplary process for logging trace records when a branch instruction is executed with trap on branch using pre and post handlers from the interrupt unit's perspective is depicted in accordance with a preferred embodiment of the present invention. The process begins when the processor, in response to a branch instruction being executed, sends a signal to the interrupt unit indicating that a branch instruction is executed (step **602**). The interrupt unit, such as interrupt unit **304** in **Figure 3**, then notifies the processor to execute instructions from the pre handler routine (step **604**). Next, the branch instruction is executed to take a branch (step **606**). Consequently, the interrupt unit notifies the processor to execute instructions from the post handler to record trace data (step **610**) prior to returning to normal execution (step **612**), the process terminating thereafter.

Thus, the present invention provides a solution of recording performance data without modifying application and/or system code at run time. Pre and post handlers are provided to process performance information in various ways. One example may be by accumulating values

of events that occurred to be used by a performance analysis tool. In addition, the pre and post handler may record trace data with "trap on branch" conditions when a branch instruction is executed. The pre and post handler may also be used to record precise performance data for events occurring before entering and immediately after exiting an interrupt handler. The recorded values provide engineers with a tool to separate events that occur during normal execution of the system from events that occur when the system is interrupted, in order to better optimize the system.

It is important to note that while the present invention has been described in the context of a fully functioning data processing system, those of ordinary skill in the art will appreciate that the processes of the present invention are capable of being distributed in the form of a computer readable medium of instructions and a variety of forms and that the present invention applies equally regardless of the particular type of signal bearing media actually used to carry out the distribution. Examples of computer readable media include recordable-type media, such as a floppy disk, a hard disk drive, a RAM, CD-ROMs, DVD-ROMs, and transmission-type media, such as digital and analog communications links, wired or wireless communications links using transmission forms, such as, for example, radio frequency and light wave transmissions. The computer readable media may take the form of coded formats that are decoded for actual use in a particular data processing system.

The description of the present invention has been presented for purposes of illustration and description, and is not intended to be exhaustive or limited to the invention in the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art. The embodiment was chosen and described in order to best explain the principles of the invention, the practical application, and to enable others of ordinary skill in the art to understand the invention for various embodiments with various modifications as are suited to the particular use contemplated.